

Credit Risk Modelling For Dummies: But With Fewer Dummies

S. J. Curran

Overview

Standard credit risk analysis utilises scorecards which are built using only datasets with categorical variables. This requires the continuous numerical features to be fine-classed (grouped into discrete sets) and converted to dummy variables. Although the point of the scorecard is to present the model in a simple way, this practice requires much convoluted pre-processing of the data, which greatly bloats the size of the dataset and makes it more susceptible to containing errors. Most importantly though, I find that, by retaining the numerical features, the predictive power of the data is greatly improved, with a Gini coefficient of 0.98 (cf. 0.40 with fine-classing) and Kolmogorov-Smirnov statistic of $KS = 0.97$ (cf. 0.30).

Although all of the processing was done in the python scripts mentioned in the report, a simplified run-through is included as the Jupyter notebooks [PP.ipynb](#) and [ML.ipynb](#)

Contents

1	Introduction	1
2	Pre-processing	1
2.1	Initial pre-processing	1
2.1.1	The data	1
2.1.2	Times and dates	2
2.1.3	Further trimming	4
2.1.4	Imputing the remaining missing values	5
2.1.5	Formatting the target variable	6
2.2	Coarse classing of categorical values	11
2.2.1	Grade	11
2.2.2	Home ownership	13
2.2.3	Verification status	13

2.2.4	Payment plan	14
2.2.5	Purpose	14
2.2.6	Address state	15
2.2.7	Initial list status	17
3	Machine Learning	18
3.1	Initial models and tests	18
3.2	Model optimisation	19
3.3	Feature importance	20
4	Results	23
4.1	ROC curve	23
4.2	Gini coefficient	24
4.3	Kolmogorov-Smirnov score	24
4.4	Using the Udemmy Target Classification	25
4.5	IVs of the Most Important Features	27

1 Introduction

Credit risk addresses the potential loss a lender or investor may face if a borrower or debtor fails to repay a loan or meet their financial obligations. It is a critical consideration for financial institutions, as a high level of credit risk can lead to significant financial losses and even bankruptcy. Accurately assessing credit risk is essential for lenders to make informed decisions about whether to extend credit and the terms of the loan.

Coarse and fine classing are methods used to assess credit risk. *Coarse classing* involves grouping borrowers or debtors into broad categories based on their creditworthiness, such as high, medium, or low risk. *Fine classing*, on the other hand, involves a more detailed analysis of credit risk, using a more nuanced approach to identify different levels of creditworthiness within each broad category.

Classing is used as credit regulators require statistical models for estimating credit risk to be presented in a simple way (the scorecard). This is based upon the coefficients in the *probability of default* model and must only contain only dummy variables as predictive features. However, fine classing (which generates typically 20 – 50 features) can lead quickly to a very bloated dataset. This requires more processing memory and is effectively an unnecessary degradation of any continuous (non-discrete data), while making the pre-processing of data more complex and thus more prone to errors.

Here I demonstrate that not fine classing can deliver a significantly greater predictive power¹, while simplifying the pre-processing of the data.

2 Pre-processing

2.1 Initial pre-processing

2.1.1 The data

The dataset `loan_data_200_2014.csv` (available from [Kaggle](#)) contains all available data for more than 800 000 consumer loans issued from 2007 to 2015 by LendingClub, a large U.S. peer to peer lending company.

Running `pre-process_1.py`, it is seen that there are 466285 rows x 75 columns. Also, it is evident that there are a lot of missing values (NaN) and a lot of feature columns which are completely empty, e.g.

```
54 annual_inc_joint          0 non-null    float64
55 dti_joint                 0 non-null    float64
56 verification_status_joint 0 non-null    float64
```

To start with, I drop the features with > 200 000 missing values (a large fraction of the data). These are

```
desc has 340302 missing values
```

¹Compared to Udemy's [Credit Risk Modelling in Python](#) and [Credit Risk Modelling \(Part II\)](#).

```

mths_since_last_delinq has 250351 missing values
mths_since_last_record has 403647 missing values
next_pymnt_d has 227214 missing values
mths_since_last_major_derog has 367311 missing values
annual_inc_joint has 466285 missing values
dti_joint has 466285 missing values
verification_status_joint has 466285 missing values
open_acc_6m has 466285 missing values
open_il_6m has 466285 missing values
open_il_12m has 466285 missing values
open_il_24m has 466285 missing values
mths_since_rcnt_il has 466285 missing values
total_bal_il has 466285 missing values
il_util has 466285 missing values
open_rv_12m has 466285 missing values
open_rv_24m has 466285 missing values
max_bal_bc has 466285 missing values
all_util has 466285 missing values
inq_fi has 466285 missing values
total_cu_tl has 466285 missing values
inq_last_12m has 466285 missing values

```

The extraneous fields – Unnamed: 0 and member_id are also removed. At this stage, id (a unique ID for the loan) is retained as this will be useful in recombining the data if this is split (e.g. Sect. 2.1.4). After removal there are 50 feature columns remaining.

2.1.2 Times and dates

Following the [Udemy](#) analysis, the next step is to covert the *employment length* (emp_length) and *term of the loan* (term) to numeric values. That is

```

['10+ years' '< 1 year' '1 year' '3 years' '8 years' '9 years' '4 years'
 '5 years' '6 years' '2 years' '7 years' nan]

[' 36 months' ' 60 months']

```

which become

```

['10' '0' '1' '3' '8' '9' '4' '5' '6' '2' '7' nan]
[36 60]

```

In order to limit empty space between the columns when viewing the data-frame, I have also shortened the feature name emp_length to EM_L.

The *earliest credit line* (earliest_cr_line_date) field also has to be converted into a useful format, and using

```
pd.to_datetime(df['earliest_cr_line'],
               format = '%b-%y')
```

gives the second column to the right from the original first. I also shorten earliest_cr_line_date to ECL.

However, looking at the converted dates, the latest is over 40 years in the future! While this suggests super predictive power it is probably inadvisable to rely on this. As the UdeMy course suggests, this is due to the built-in time scale starting from 1970 (it actually looks like the start of 1969).

```
0      Jan-85      0      1985-01-01
1      Apr-99      1      1999-04-01
2      Nov-01      2      2001-11-01
3      Feb-96      3      1996-02-01
4      Jan-96      4      1996-01-01
...
466280 Apr-03      466280 2003-04-01
466281 Jun-97      466281 1997-06-01
466282 Dec-01      466282 2001-12-01
466283 Feb-03      466283 2003-02-01
466284 Feb-00      466284 2000-02-01

count      466256
unique      664
top      2000-10-01 00:00:00
freq      3674
first      1969-01-01 00:00:00
last      2068-12-01 00:00:00
Name: ECL, dtype: object
```

Looking at the latest earliest_cr_line_date (ECL) before today

```
from datetime import date
today = np.datetime64(date.today()); print(today)
temp = df[df['ECL'] < today]; print(temp['ECL'].describe())
```

gives 2011-11-01 as the latest ECL date.

The earliest credit line is then converted to months since this date (UdeMy uses December 2017), giving the output to the right.

The error in the dates seen above is apparent from the minimum number of months since the earliest credit line.

```
count      466256.000000
mean      166.492177
std       93.976747
min      -685.024333
max       513.981807
```

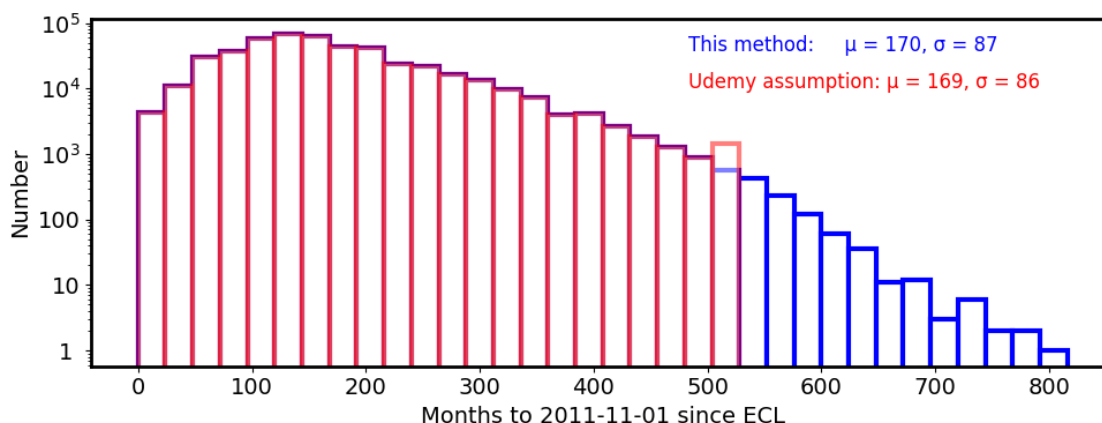
The [UdeMy](#) course deals with this by substituting the negative values for the maximum observed positive difference, with the justification “*Even if we don’t calculate the exact number of months that have passed since the earliest credit line was issued for those issued in the 60s, we put a very large value and we still get pretty close to the real picture.*”

However, it is arguable whether these values are *very large*, as well as the potential of unnecessarily losing crucial information. By splitting the date string and subtracting 100 from the year for which the number of months are negative, we can obtain the correct number of months since the earliest credit line.

```
count      466256.000000      count      466256.000000
mean      169.395094      mean      169.500887
std       86.461715      std       86.930703
```

min	0.000000	min	0.000000
max	513.981807	max	814.012608

From the mean values and the plot below, it is seen that the Udeemy assumption should have minimum impact, although this could be an issue if working with a significant fraction of older (pre-1970) loans. So while the assumption is easier to implement, it may not always be warranted.



This procedure is repeated to get the months since the loan issue date (`issue_d`). This spans 2007-06-01 to 2014-12-01 and so no conversion of pre-1970 dates is required. This is also the case for `last_pymnt_d` (2007-12-01 to 2016-01-01) and `last_credit_pull_d` (2007-12-01 to 2016-01-01), which are also converted.

```
count      466256
unique           91
top    2014-10-01 00:00:00
freq      38782
first    2007-06-01 00:00:00
last     2014-12-01 00:00:00
Name: i_date, dtype: object
```

2.1.3 Further trimming

Before dealing with the categorical features, the data which appear to be of limited value are further trimmed:

- `url` – as these are unique to each customer²
- `zip_code` – which are only partial (e.g. 860xx) and we will use the *address state* (`addr_state`)
- `sub_grade` – as credit worthiness grade (`grade`) will be used, and perhaps coarse classed (Sect. 2.2.1) so that higher resolution values will not be needed
- `application_type` – which only has a single value (namely INDIVIDUAL)

Revisiting the missing values

²Of the form [https://www.lendingclub.com/browse/loanDetail.action?loan_id= ...](https://www.lendingclub.com/browse/loanDetail.action?loan_id=...)

```

emp_title has 27541 missing values
title has 20 missing values
revol_util has 305 missing values
collections_12_mths_ex_med has 115 missing values
tot_coll_amt has 70130 missing values
tot_cur_bal has 70130 missing values
total_rev_hi_lim has 70130 missing values
EM_L has 20983 missing values

```

the employment length (EM_L) could be important, as well as the borrower's job title (emp_title), so I initially removed only those with > 30 000 missing values. Of the remaining missing values:

- emp_title has 205 271 unique values, but these are categorical, which would require too many dummy variables, so this is *removed*.
- title has 63015 unique values which are also categorical, so this is *removed*.
- revol_util has 1270 unique values which are numerical, so this will be *imputed*.
- collections_12_mths_ex_med has 10 unique values which are numerical, so this will be *imputed*.
- EM_L has 12 unique values which are numerical, so this will be *imputed*.

Which now leaves 41 features in the data.

2.1.4 Imputing the remaining missing values

There are various methods available to impute missing data, with the most common being *simple imputation*, where the missing feature is replaced with either a constant, the mean, the median or the most frequently occurring value. Since this is basically just using an assumed value, I opted for *multivariate (multiple) imputation*, where machine learning is used to estimate the missing values from the other features.

Bearing in mind the caveats that the employment length EM_L contains both lower (10+ years) and upper limits (< 1 year), I summarise the effects of the imputation below.

```

----- BEFORE IMPUTATION -----
For revol_util, n = 465534, mean = 56.181 +/- 23.730
For collections_12_mths_ex_med, n = 465724, mean = 0.009 +/- 0.109
For EM_L, n = 444856, mean = 5.994 +/- 3.627

----- AFTER IMPUTATION -----
For revol_util, n = 465839, mean = 56.186 +/- 23.725
For collections_12_mths_ex_med, n = 465839, mean = 0.009 +/- 0.109
For EM_L, n = 465839, mean = 6.019 +/- 3.555

```

2.1.5 Formatting the target variable

The target variable is the `loan_status`, which has the unique values

```
'Fully Paid'           'Late (31-120 days)'
'Charged Off'          'In Grace Period'
'Late (16-30 days)'    'Does not meet the credit policy. Status:Fully Paid'
'Current'              'Does not meet the credit policy. Status:Charged Off'
'Default'
```

of which, e.g. Fully Paid, Current and In Grace Period are considered good loans and Charged Off and Default as bad.

In order to classify these, I calculate the *debt-to-income ratio* (DTIR) for each category.³ To get the debt, I subtracted the *principal received to date* (`total_rec_prncp`) from the *loan amount* (`loan_amnt`). In increasing debt-to-income ratio, the results are

	loan_status	number	mean DTIR	standard error
0	Fully Paid	184724	0.0008	0.0
7	Does not meet the credit policy. Status:Fully ...	1961	0.0027	0.0004
8	Does not meet the credit policy. Status:Charge...	746	0.1194	0.0036
2	Current	224207	0.1283	0.0002
5	In Grace Period	3146	0.1399	0.0016
6	Late (16-30 days)	1218	0.1411	0.0026
4	Late (31-120 days)	6900	0.1587	0.0011
3	Default	832	0.1743	0.0034
1	Charged Off	42105	0.1833	0.0005

I flagged all statuses above Late (16-30 days), or DTIR $\lesssim 0.15$, as good/non-defaulted (`good = 1`) and those below, or DTIR $\gtrsim 0.15$, as bad/defaulted (`good = 0`) loans.

This results in 416 002 `good = 1` loans and 49837 `good = 0`, which differ slightly from the Udeemy numbers (see Sect. 4.4), which are from an unspecified regression model.

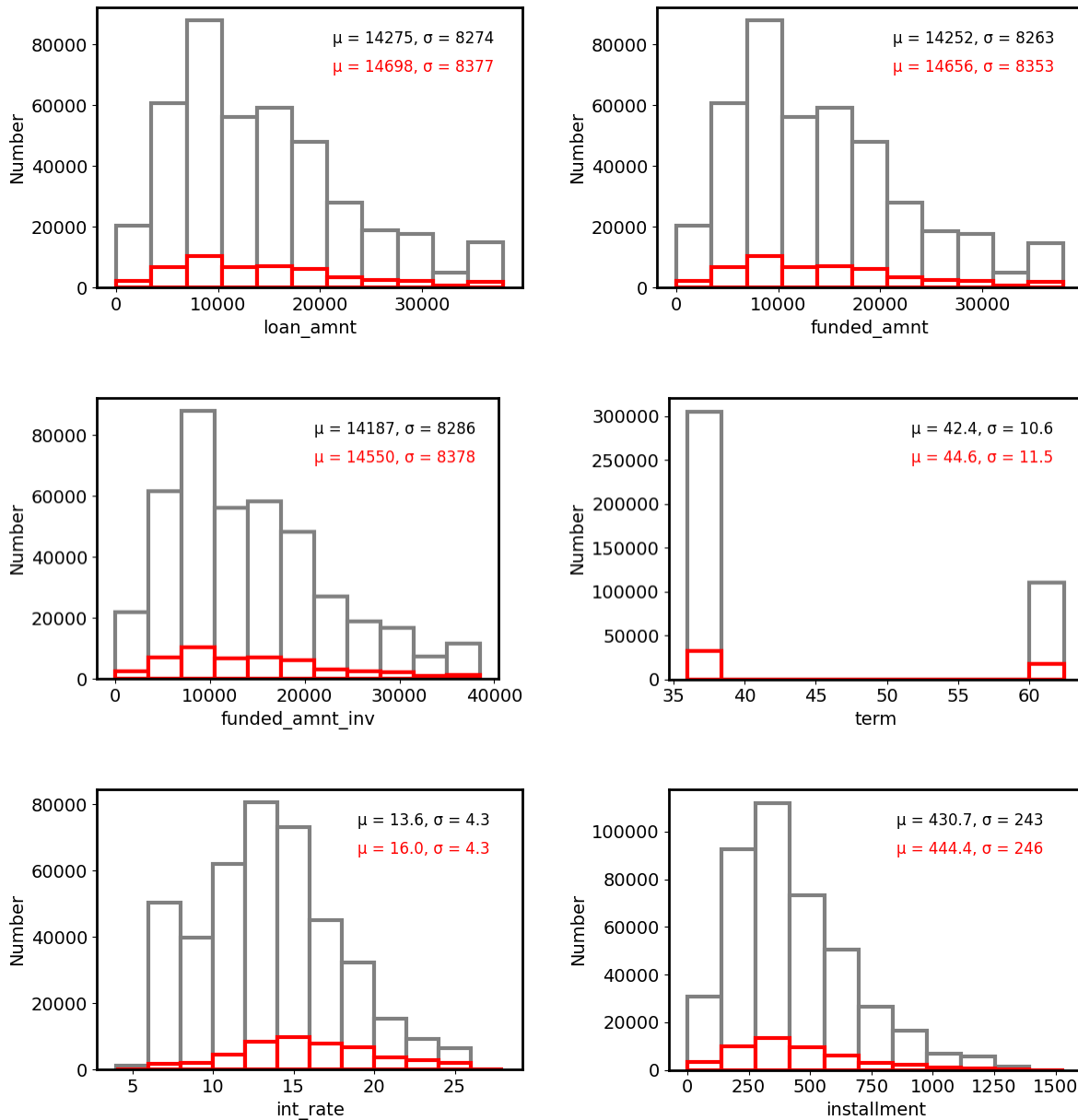
	loan_status	good
0	Fully Paid	1
1	Charged Off	0
2	Fully Paid	1
...
465836	Current	1
465837	Fully Paid	1
465838	Current	1

Since this completes the first phase of the pre-processing and the code is becoming cumbersome, the currently processed data are saved to `loan_data_2007_2014_1.csv`.

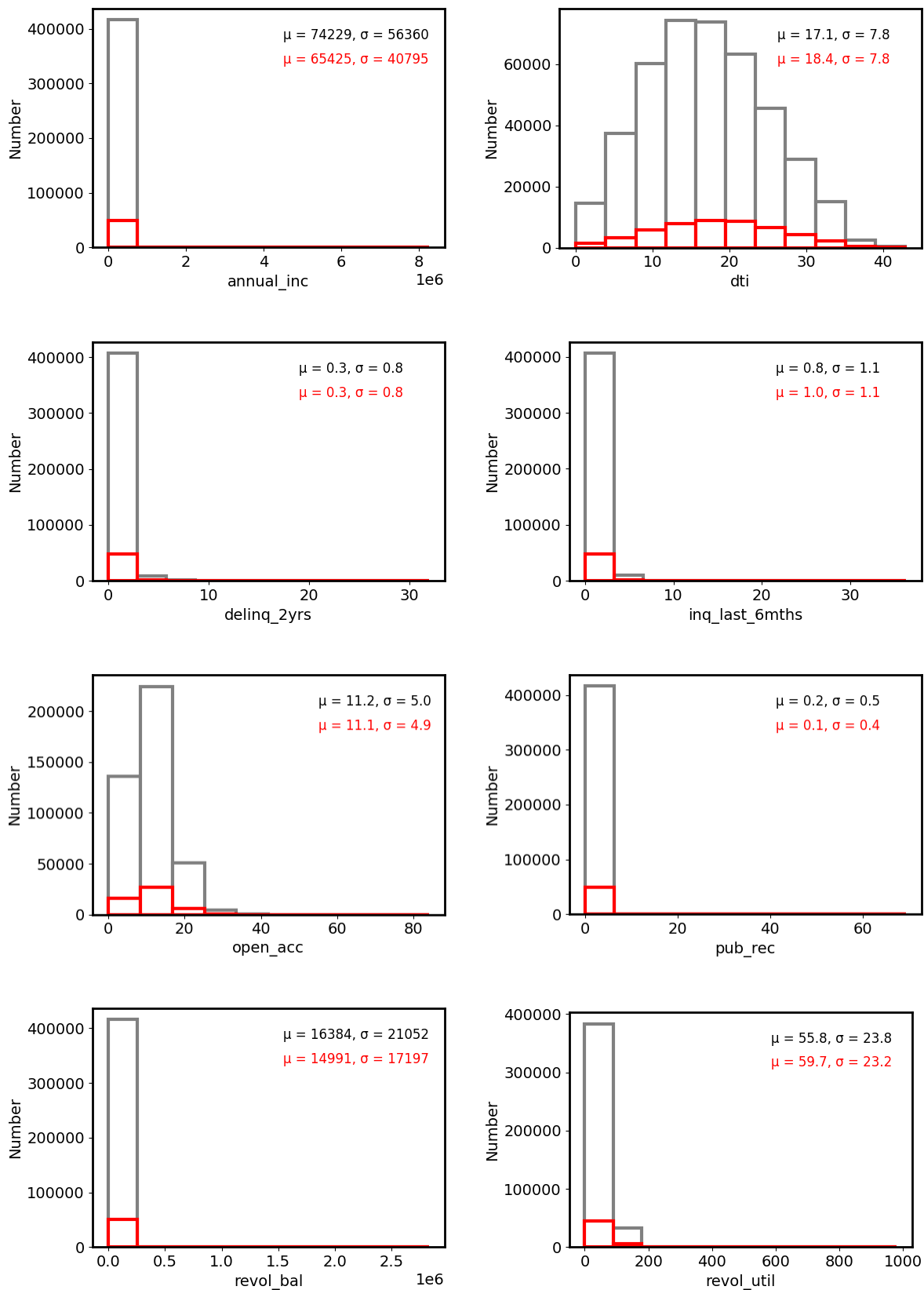
This file is read into `feature_histos.py` to show the distributions for each of the numeric features according to good (black) and bad (red histogram) loans. From these, the good and bad loans mostly overlap, although

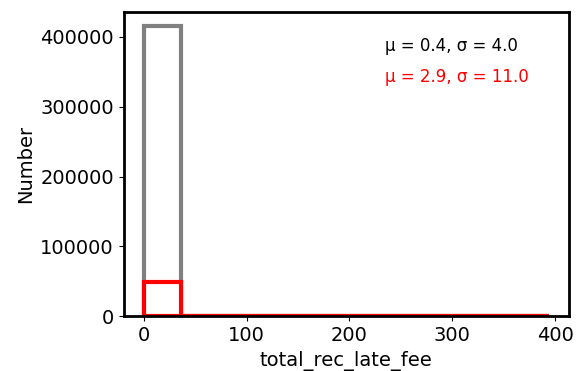
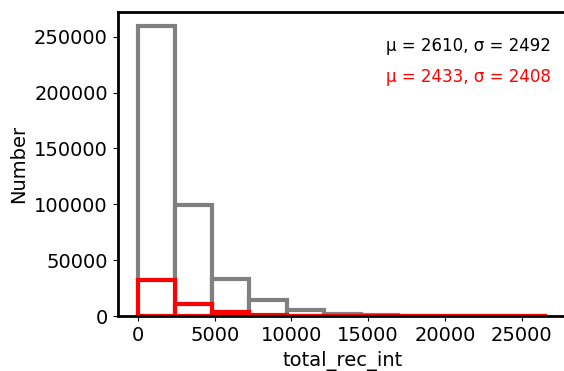
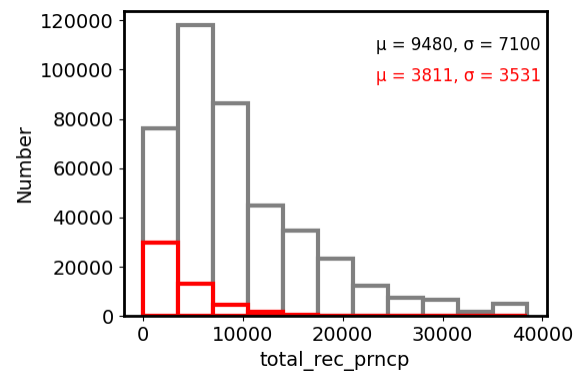
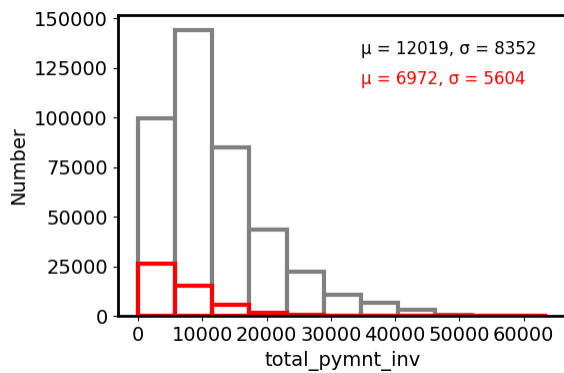
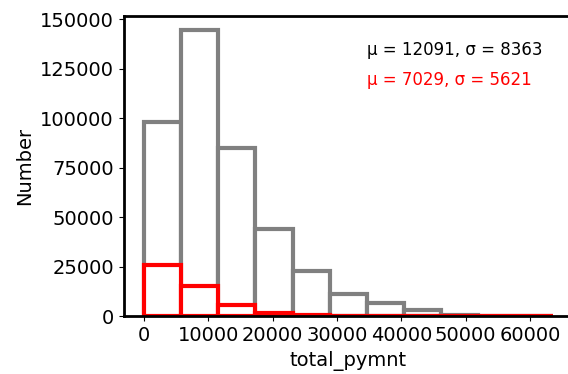
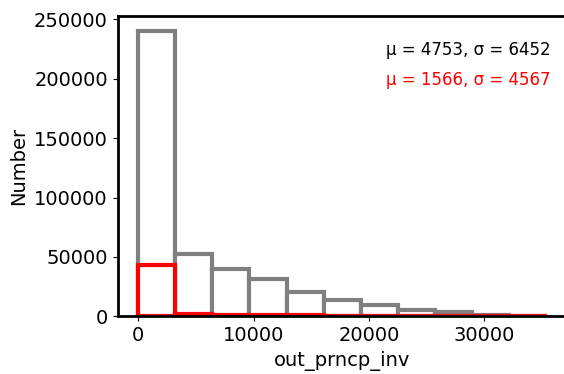
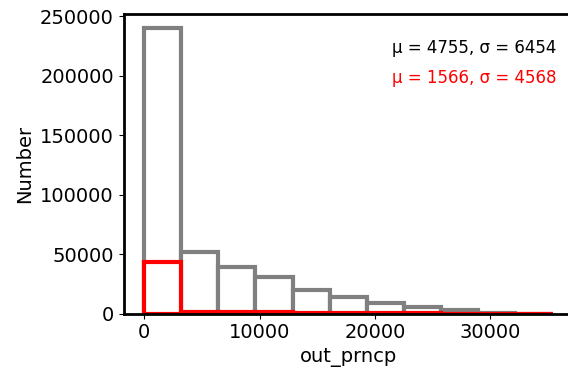
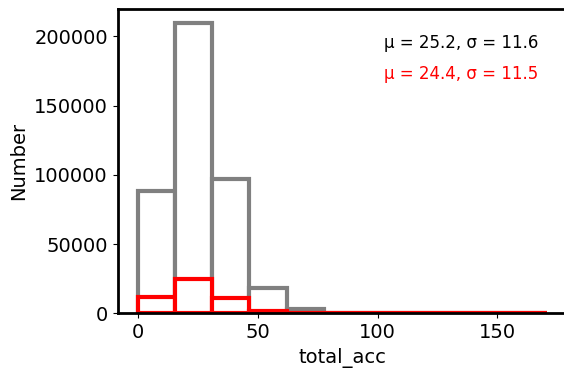
³Note that the feature `dti` is not the debt-to-income ratio, but, according to [Kaggle](#), a ratio of the borrower's total monthly debt payments on the total debt obligation.

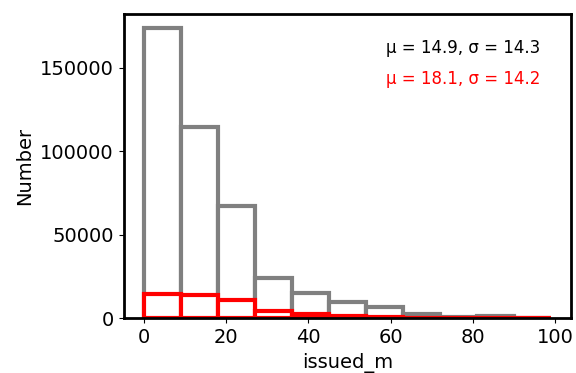
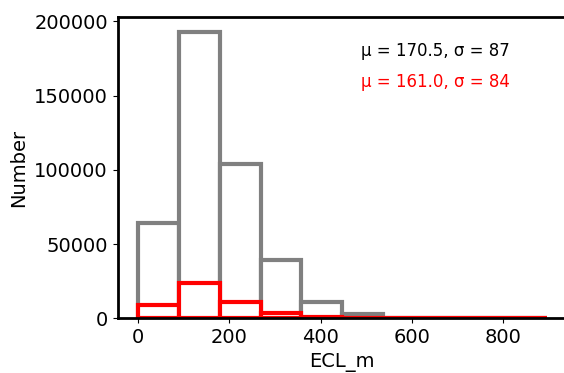
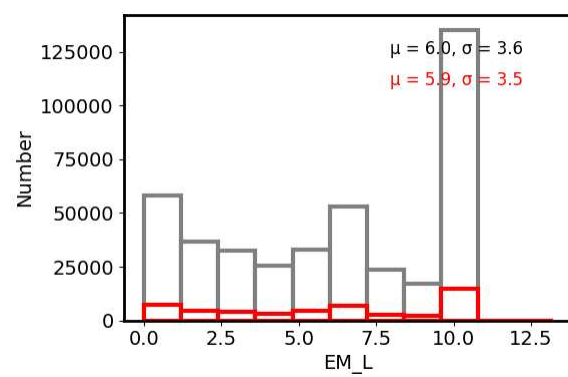
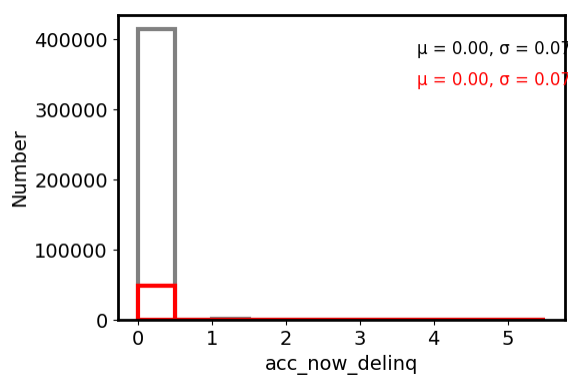
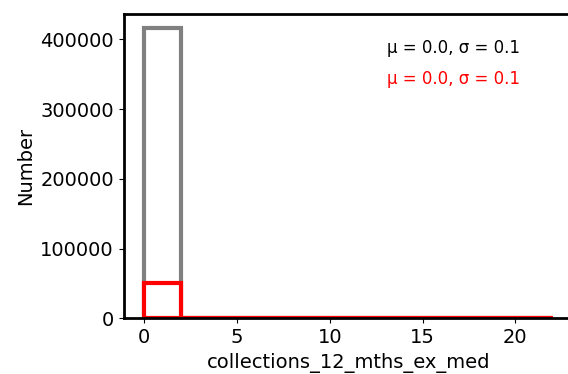
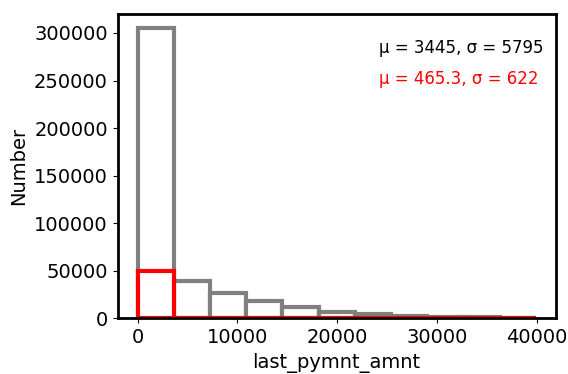
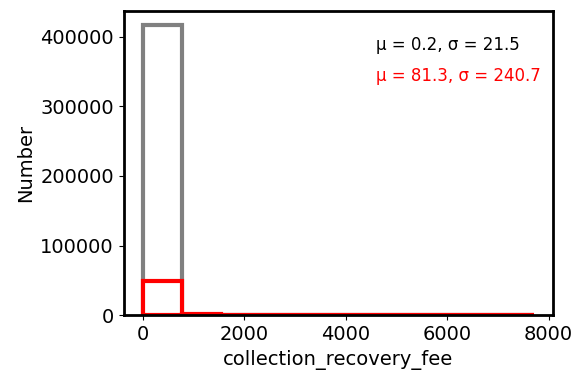
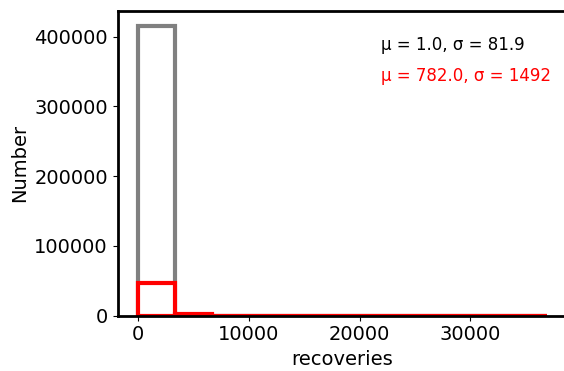
recoveries, collection_recovery_fee, LP_m, debt and, not surprisingly⁴, DTIR appear distinctive, the standard deviations in the data are very wide. This demonstrates that bad loans would be difficult to distinguish based upon the distributions alone.

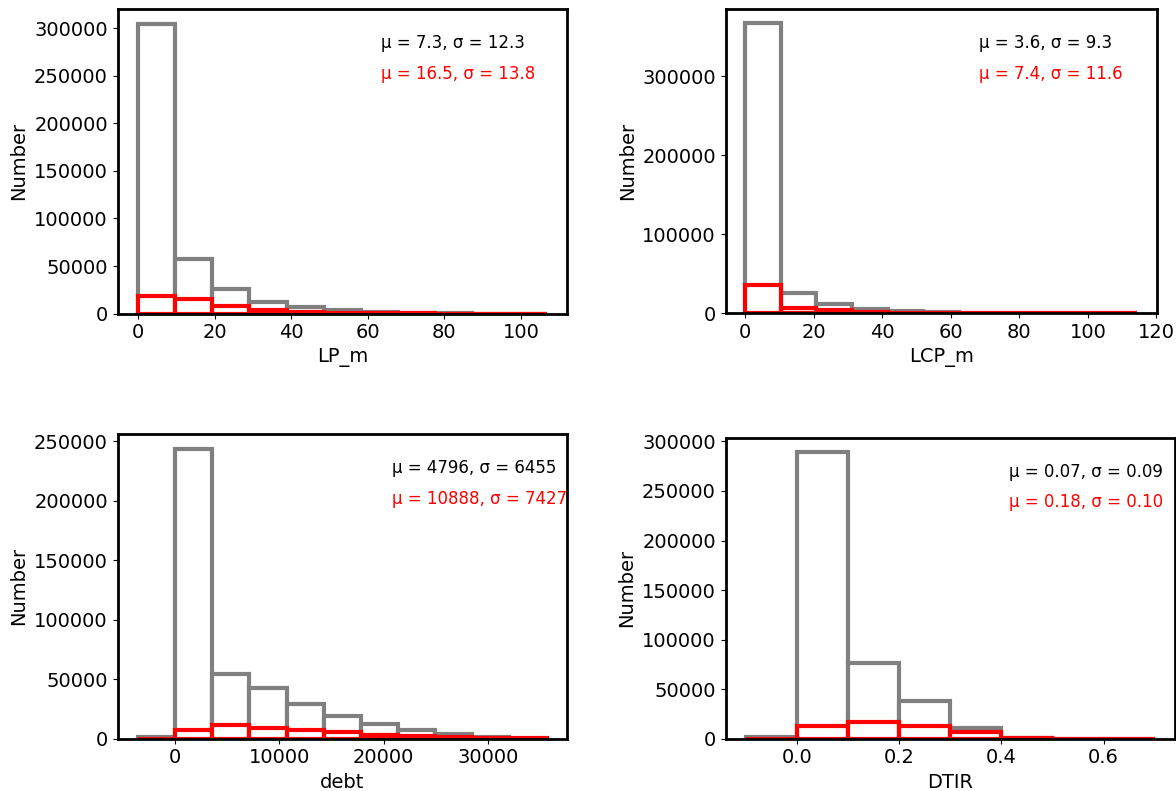


⁴Since this was used to classify the loan as being good or bad.









2.2 Coarse classing of categorical values

Although, unlike the standard practice (e.g. [Udemy](#)), I will not convert the continuous variables to dummy variables, this should be done for the categorical values in order to fully utilise the available data. I will then assess the feature's explanatory power (information value) with respect to the borrower classification.

The remaining categorical fields are

```
['grade', 'home_ownership', 'verification_status', 'pymnt_plan', 'purpose',
 'addr_state', 'initial_list_status'],
```

which are worked through one-by-one in `pre-process_2.py`, below.

2.2.1 Grade

The grade feature, which represents an external grade showing the credit worthiness of the borrower, ranging from A (the highest) to G (the lowest), has only seven unique values.

Here and for the following categories, as per the standard practice, we wish to know how relevant the feature

is to the target variable. For this the *weight of evidence* for each value of the feature is used,

$$WoE_i \equiv \ln \left(\frac{\%(y=1)_i}{\%(y=0)_i} \right) = \ln \left(\frac{\% \text{ good}_i}{\% \text{ bad}_i} \right),$$

which is the natural logarithm of the ratio of proportion of observations in the first target category to those in the second.

Multiplying the *WoE* by the difference in the proportions and summing, gives the *information value*

$$IV = \sum_{i=1}^k \left[(\% \text{ good} - \% \text{ bad}) \times \ln \left(\frac{\% \text{ good}}{\% \text{ bad}} \right) \right],$$

which quantifies how much information the feature contributes to explaining the target. The information value ranges from $IV = 0$ to 1 according to its predictive power (summarised in the table).

Information value	Predictive power
$IV < 0.02$	None
$0.02 < IV < 0.1$	Weak
$0.1 < IV < 0.3$	Medium
$0.3 < IV < 0.5$	Strong
$IV > 0.5$	Suspect

Proceeding with the grade feature, in increasing *WoE*

```
n_obs - number of observations for this grade
%n_obs - percentage of total number of observations
n_good - number of good = 1 observations for this grade (bad loans are flagged 'good = 0')
%good - percentage of good = 1 observations for this grade
%n_good - percentage of total number
```

	grade	n_obs	%n_obs	n_good	n_bad	%good	%bad	%n_good	%n_bad	WoE	IV
0	G	3316	0.71	2470	846	74.49	25.51	0.59	1.70	-1.06	0.011766
1	F	13205	2.83	10129	3076	76.71	23.29	2.43	6.17	-0.93	0.034782
2	E	35679	7.66	28945	6734	81.13	18.87	6.96	13.51	-0.66	0.043230
3	D	76784	16.48	65228	11556	84.95	15.05	15.68	23.19	-0.39	0.029289
4	C	125186	26.87	111042	14144	88.70	11.30	26.69	28.38	-0.06	0.001014
5	B	136848	29.38	126233	10615	92.24	7.76	30.34	21.30	0.35	0.031640
6	A	74821	16.06	71955	2866	96.17	3.83	17.30	5.75	1.10	0.127050

```
=====
For 'grade' feature, sum of IVs = 0.2788: 0.1 < IV < 0.3 - medium predictive power
=====
```

The predictive power of the grade feature is close to the medium/strong threshold.

Since there are only seven different values, each with a large number of observations, the dummy variables can be one-hot-encoded directly, thus not requiring coarse classing, giving, for example,

	good	grade_A	grade_B	grade_C	grade_D	grade_E	grade_F	grade_G
0	1	0	1	0	0	0	0	0
1	0	0	0	1	0	0	0	0

```

2          1          0          0          1          0          0          0          0
...        ...        ...        ...        ...        ...        ...        ...        ...
465837     1          1          0          0          0          0          0          0
465838     1          0          0          0          1          0          0          0

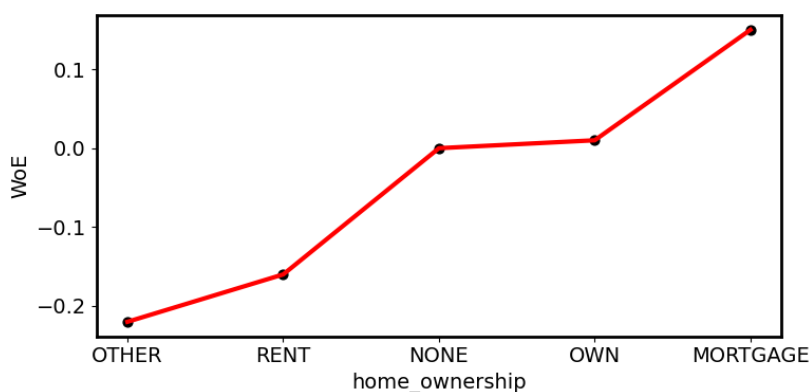
```

2.2.2 Home ownership

Since the *WoE* and *IV* calculations are in a function (in `pre-process_2.py`), it is trivial to get these metrics for the remaining features. Continuing with `home_ownership`

	home_ownership	n_obs	%n_obs	n_good	n_bad	%good	%bad	%n_good	%n_bad	WoE	IV
0	OTHER	182	0.04	155	27	85.16	14.84	0.04	0.05	-0.22	0.000022
1	RENT	188208	40.40	165019	23189	87.68	12.32	39.67	46.53	-0.16	0.010976
2	NONE	46	0.01	39	7	84.78	15.22	0.01	0.01	0.00	0.000000
3	OWN	41654	8.94	37231	4423	89.38	10.62	8.95	8.87	0.01	0.000008
4	MORTGAGE	235748	50.61	213557	22191	90.59	9.41	51.34	44.53	0.14	0.009534
5	ANY	1	0.00	1	0	100.00	0.00	0.00	0.00	NaN	NaN

For 'home_ownership' feature, sum of IVs = 0.0205: $0.02 < IV < 0.1$ - weak predictive power



If using this feature, given the low *IV*, the small numbers (`n_obs`) for OTHER, NONE and ANY means that it is not sensible to one-hot-encode these. Therefore, OTHER is merged with the nearest populous value with a similar *WoE*, RENT. Likewise, NONE and ANY is merged with OWN.

After merging (and renaming for neatness), we are left with the fields `HO_MORTGAGE` `HO_RENT+` `HO_OWN+`, where the '+' designates the inclusion of other groups.

2.2.3 Verification status

There is no information regarding the `verification_status` on either Udemy's [Credit Risk Modelling in Python](#) nor [Kaggle](#).

Running the `woe` function in `pre-process_2.py`, shows that there are only three unique values, each of which has a large, and similar, number of observations.

```

      VS  n_obs  %n_obs  n_good  n_bad  %good  %bad  %n_good  %n_bad  WoE      IV
0      Verified 167920  36.05 146802 21118  87.42 12.58   35.29  42.37 -0.18  0.012744
1 Source Verified 149862  32.17 134444 15418  89.71 10.29   32.32  30.94  0.04  0.000552
2   Not Verified 148057  31.78 134756 13301  91.02  8.98   32.39  26.69  0.19  0.010830
=====
For 'VS' feature, sum of IVs = 0.0241: 0.02 < IV < 0.1 - weak predictive power
=====

```

Therefore, there is no need to coarse class this feature, which are therefore converted to dummy variables directly.

2.2.4 Payment plan

This flags whether a payment plan has been put in place for the loan.

```

PP  n_obs  %n_obs  n_good  n_bad  %good  %bad  %n_good  %n_bad  WoE  IV
0  y      9      0.0      5      4  55.56  44.44      0.0   0.01 -inf  inf
1  n 465830 100.0 415997 49833  89.30 10.70     100.0  99.99  0.0  0.0
=====
For 'PP' feature, sum of IVs = inf: IV > 0.5 - something fishy going on
=====

```

There are just two values, with the number of PP = y being so small that the feature is almost exclusively PP = n. This highly unbalanced feature is therefore removed.

2.2.5 Purpose

This is the purpose of the loan.

```

      P      n_obs  %n_obs  n_good  n_bad  %good  %bad  %n_good  %n_bad  WoE      IV
small_business  6993  1.50   5572  1421  79.68  20.32   1.34  2.85 -0.75  0.011325
moving         2990  0.64   2551  439   85.32  14.68   0.61  0.88 -0.37  0.000999
renewable_energy 349  0.07    298  51   85.39  14.61   0.07  0.10 -0.36  0.000108
house          2264  0.49   1962  302   86.66  13.34   0.47  0.61 -0.26  0.000364
other          23607 5.07  20507 3100   86.87  13.13   4.93  6.22 -0.23  0.002967
educational     419  0.09    366  53   87.35  12.65   0.09  0.11 -0.20  0.000040
medical         4588 0.98   4007  581   87.34  12.66   0.96  1.17 -0.20  0.000420
vacation        2484 0.53   2200  284   88.57  11.43   0.53  0.57 -0.07  0.000028
debt_consolidation 273984 58.82 243389 30595  88.83  11.17  58.51 61.39 -0.05  0.001440
wedding         2331 0.50   2071  260   88.85  11.15   0.50  0.52 -0.04  0.000008
home_improvement 26510 5.69  24001 2509   90.54   9.46   5.77  5.03  0.14  0.001036
major_purchase  9813 2.11   8911  902   90.81   9.19   2.14  1.81  0.17  0.000561
car             5389 1.16   4929  460   91.46   8.54   1.18  0.92  0.25  0.000650

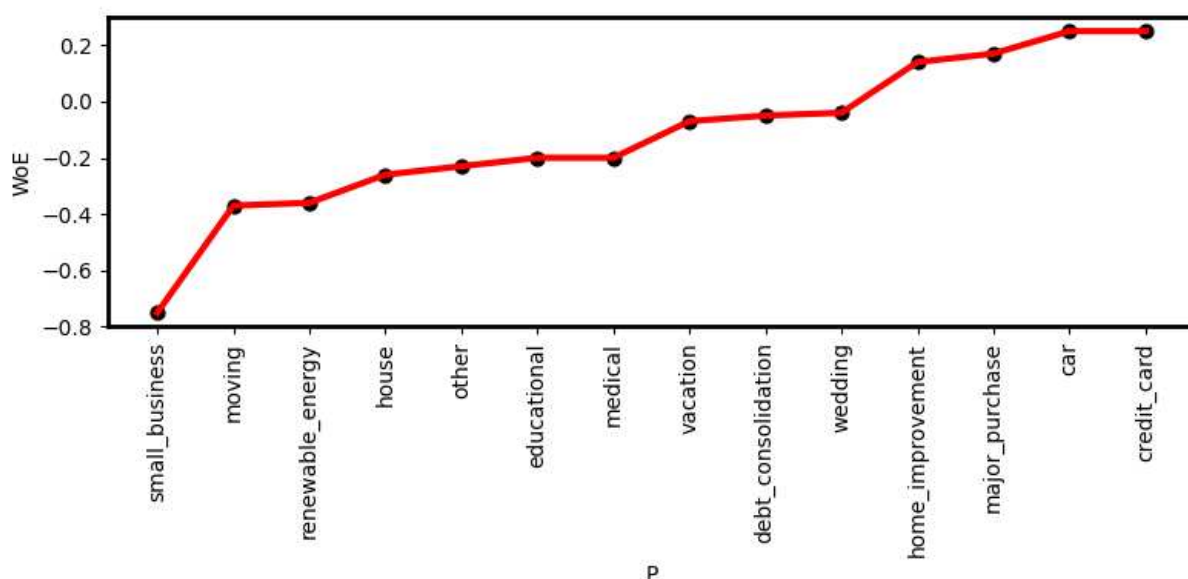
```



```

credit_card 104118 22.35 95238 8880 91.47 8.53 22.89 17.82 0.25 0.012675
=====
For 'P' feature, sum of IVs = 0.0326: 0.02 < IV < 0.1 - weak predictive power
=====

```



There are some low numbers of observations (e.g. `renewable_energy`), which could be merged with the other values (as in Sect. 2.2.2), but given that there are 14 unique values, these should be bundled into coarser classes in any case.

With the guidance of the *WoE* values in the above table and the plot, the dummies are combined as

```

comb_dummy('P', ['small_business', 'moving', 'renewable_energy'])
comb_dummy('P', ['other', 'house', 'educational', 'medical'])
comb_dummy('P', ['debt_consolidation', 'vacation', 'wedding'])
comb_dummy('P', ['home_improvement', 'major_purchase'])
comb_dummy('P', ['credit_card', 'car']);

```

leaving the features

```

P_small_business+ P_other+ P_debt_consolidation+ P_home_improvement+ P_credit_card+

```

2.2.6 Address state

	AS	n_obs	%n_obs	n_good	n_bad	%good	%bad	%n_good	%n_bad	WoE	IV
0	NE	14	0.00	9	5	64.29	35.71	0.00	0.01	-inf	inf
1	NV	6508	1.40	5641	867	86.68	13.32	1.36	1.74	-0.25	0.000950
2	HI	2485	0.53	2175	310	87.53	12.47	0.52	0.62	-0.18	0.000180

3	FL	31609	6.79	27738	3871	87.75	12.25	6.67	7.77	-0.15	0.001650
4	AL	5849	1.26	5138	711	87.84	12.16	1.24	1.43	-0.14	0.000266
5	NY	40189	8.63	35585	4604	88.54	11.46	8.55	9.24	-0.08	0.000552
6	LA	5483	1.18	4853	630	88.51	11.49	1.17	1.26	-0.07	0.000063
7	OK	4111	0.88	3644	467	88.64	11.36	0.88	0.94	-0.07	0.000042
8	NM	2586	0.56	2292	294	88.63	11.37	0.55	0.59	-0.07	0.000028
9	NC	12669	2.72	11233	1436	88.67	11.33	2.70	2.88	-0.06	0.000108
10	NJ	18039	3.87	16018	2021	88.80	11.20	3.85	4.06	-0.05	0.000105
11	MD	10959	2.35	9735	1224	88.83	11.17	2.34	2.46	-0.05	0.000060
12	VA	14211	3.05	12616	1595	88.78	11.22	3.03	3.20	-0.05	0.000085
13	CA	71369	15.32	63425	7944	88.87	11.13	15.25	15.94	-0.04	0.000276
14	MO	7500	1.61	6669	831	88.92	11.08	1.60	1.67	-0.04	0.000028
15	UT	3428	0.74	3052	376	89.03	10.97	0.73	0.75	-0.03	0.000006
16	TN	5980	1.28	5323	657	89.01	10.99	1.28	1.32	-0.03	0.000012
17	MI	11542	2.48	10283	1259	89.09	10.91	2.47	2.53	-0.02	0.000012
18	AZ	10702	2.30	9534	1168	89.09	10.91	2.29	2.34	-0.02	0.000010
19	AR	3488	0.75	3108	380	89.11	10.89	0.75	0.76	-0.01	0.000001
20	SD	980	0.21	876	104	89.39	10.61	0.21	0.21	0.00	0.000000
21	RI	2049	0.44	1828	221	89.21	10.79	0.44	0.44	0.00	0.000000
22	PA	16416	3.52	14660	1756	89.30	10.70	3.52	3.52	0.00	0.000000
23	OH	15228	3.27	13605	1623	89.34	10.66	3.27	3.26	0.00	0.000000
24	KY	4433	0.95	3967	466	89.49	10.51	0.95	0.94	0.01	0.000001
25	MN	8150	1.75	7292	858	89.47	10.53	1.75	1.72	0.02	0.000006
26	IN	6523	1.40	5841	682	89.54	10.46	1.40	1.37	0.02	0.000006
27	MA	11058	2.37	9913	1145	89.65	10.35	2.38	2.30	0.03	0.000024
28	DE	1270	0.27	1139	131	89.69	10.31	0.27	0.26	0.04	0.000004
29	WA	10508	2.26	9455	1053	89.98	10.02	2.27	2.11	0.07	0.000112
30	GA	14956	3.21	13451	1505	89.94	10.06	3.23	3.02	0.07	0.000147
31	OR	5946	1.28	5357	589	90.09	9.91	1.29	1.18	0.09	0.000099
32	WI	5908	1.27	5329	579	90.20	9.80	1.28	1.16	0.10	0.000120
33	TX	36398	7.81	32999	3399	90.66	9.34	7.93	6.82	0.15	0.001665
34	CT	7196	1.54	6537	659	90.84	9.16	1.57	1.32	0.17	0.000425
35	IL	18602	3.99	16890	1712	90.80	9.20	4.06	3.44	0.17	0.001054
36	SC	5580	1.20	5077	503	90.99	9.01	1.22	1.01	0.19	0.000399
37	CO	9735	2.09	8861	874	91.02	8.98	2.13	1.75	0.20	0.000760
38	AK	1251	0.27	1143	108	91.37	8.63	0.27	0.22	0.20	0.000100
39	MT	1394	0.30	1271	123	91.18	8.82	0.31	0.25	0.22	0.000132
40	KS	4186	0.90	3817	369	91.18	8.82	0.92	0.74	0.22	0.000396
41	NH	2230	0.48	2048	182	91.84	8.16	0.49	0.37	0.28	0.000336
42	VT	904	0.19	827	77	91.48	8.52	0.20	0.15	0.29	0.000145
43	MS	1224	0.26	1124	100	91.83	8.17	0.27	0.20	0.30	0.000210
44	WV	2410	0.52	2220	190	92.12	7.88	0.53	0.38	0.33	0.000495
45	WY	1128	0.24	1045	83	92.64	7.36	0.25	0.17	0.39	0.000312
46	DC	1425	0.31	1331	94	93.40	6.60	0.32	0.19	0.52	0.000676
47	IA	14	0.00	13	1	92.86	7.14	0.00	0.00	NaN	NaN

```

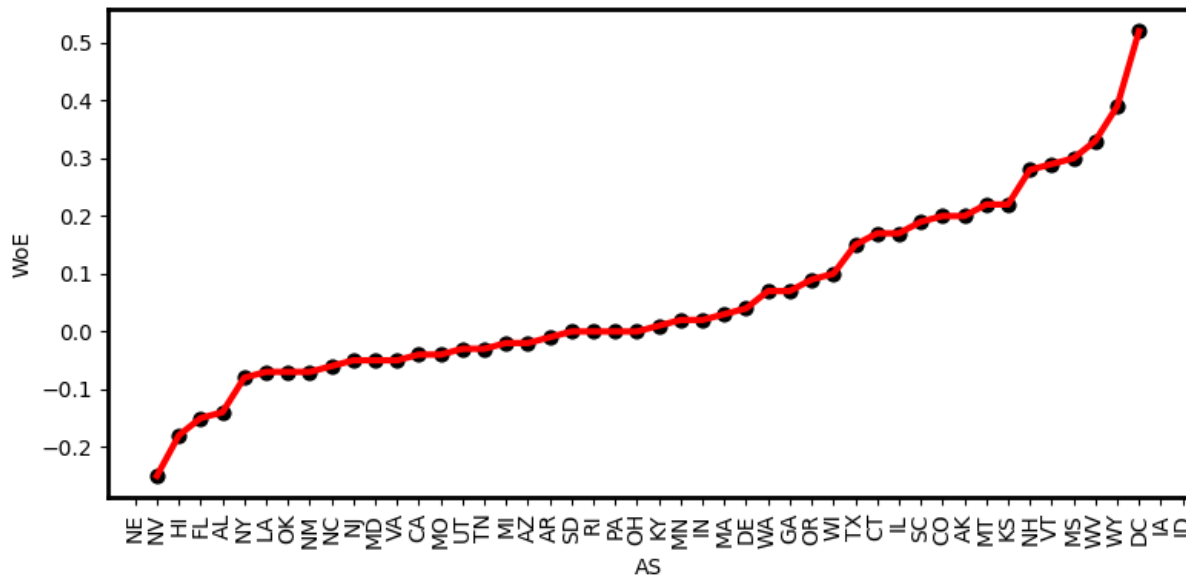
48 ID      12      0.00      11      1  91.67  8.33      0.00      0.00  NaN      NaN
49 ME       4      0.00       4      0 100.00  0.00      0.00      0.00  NaN      NaN

```

```

=====
For 'AS' feature, sum of IVs = 0.0121: IV < 0.02 - no predictive power
=====

```



Again, there are values which have a very low number of observations (e.g. NE), as well 50 different classes. These are therefore merged, according to `n_obs` and `WoE`, as

```

comb_dummy('AS', ['FL', 'NV', 'HI', 'NE', 'AL'])      AS_CA 71369      AS_WA 10508
comb_dummy('AS', ['LA', 'OK', 'NM'])                  AS_MD 10959      AS_FL+ 46465
comb_dummy('AS', ['MO', 'UT', 'TN'])                  AS_MI 11542      AS_LA+ 12180
comb_dummy('AS', ['AZ', 'AR', 'SD', 'RI'])             AS_NC 12669      AS_MO+ 16908
comb_dummy('AS', ['KY', 'MN', 'IN', 'MA', 'DE'])       AS_NJ 18039      AS_KY+ 31434
comb_dummy('AS', ['GA', 'OR', 'WI'])                  AS_NY 40189      AS_GA+ 26810
comb_dummy('AS', ['CT', 'IL'])                        AS_OH 15228      AS_CT+ 25798
comb_dummy('AS', ['SC', 'CO', 'AK', 'MT', 'KS'])       AS_PA 16416      AS_SC+ 22146
comb_dummy('AS', ['NH', 'VT', 'MS', 'WV'])             AS_TX 36398      AS_NH+ 6768
comb_dummy('AS', ['WY', 'DC', 'IA', 'ID', 'ME']);      AS_VA 14211      AS_WY+ 2583

```

where the resulting classes and `n_obs` are shown in the two right-hand columns. Arguably `AS_NH+` and `AS_WY+` could also be merged to increase the size of this class, but with $WoE = 0.28 - 0.33$, cf. ≥ 0.39 , respectively, this is left as is.

2.2.7 Initial list status

This is the initial listing status of the loan.

```

    ILS  n_obs  %n_obs  n_good  n_bad  %good  %bad  %n_good  %n_bad  WoE      IV
0   f  302666   64.97  267251  35415  88.30  11.70    64.24   71.06 -0.10  0.006820
1   w  163173   35.03  148751  14422  91.16   8.84    35.76   28.94  0.21  0.014322

```

```

=====
For 'ILS' feature, sum of IVs = 0.0211: 0.02 < IV < 0.1 - weak predictive power
=====

```

With only two values, each with a large number of observations, this can be converted directly to dummy variables.

Now that the second phase of the pre-processing is complete, the fully processed data are saved (to `loan_data_2007_2014_2.csv`).

3 Machine Learning

3.1 Initial models and tests

The standard practice is to model the *probability of default* (PD) with the logistic regression classifier. Since, it is straightforward to run other common, but different, binary classifiers in tandem with this, I include these in the modelling. That is, the data are tested with:

1. *Logistic Regression* (LR)
2. *k-Nearest Neighbour* (kNN)
3. *Support Vector Classifier* (SVC)
4. *Decision Tree Classifier* (DTC)

Each of which is described in further detail by [Curran \(2021\)](#).

The machine learning is performed in `ML.py`, where the imbalance in the data has to be addressed. Specifically, there are 49 782 bad loans compared to $465\,839 - 49\,837 = 416\,057$ good loans, and so from raw statistics alone we can ascertain a $100 \times 416\,057 / 465\,839 = 89.3\%$ probability of the loan being good, although this would have no predictive power. Therefore 49 837 of the good loan from the pool of 416 057 are randomly selected for the binary testing.

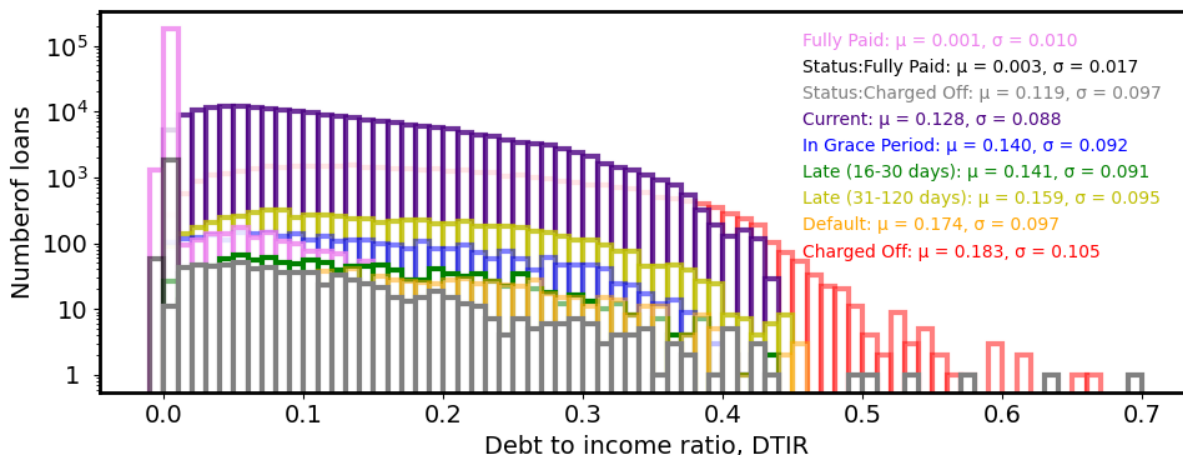
Training on 80% and testing on 20% of the sample, using (close to) the default standard parameters (`n_neighbors = 10` and `max_depth = 5`), high scores are obtained, especially for the decision tree classifier.

```

For a 0.2 test fraction (79651 train & 19913 test) LR score = 92.418
For a 0.2 test fraction (79651 train & 19913 test) KNN score = 80.765
For a 0.2 test fraction (79651 train & 19913 test) SVC score = 93.851
For a 0.2 test fraction (79651 train & 19913 test) DTC score = 98.463

```

Such high scores may not be unexpected since the target was defined via the `loan_status`, i.e. `good = 1` where `DTIR < 0.15` and `good = 0` where `DTIR > 0.15` (Sect. 2.1.5).

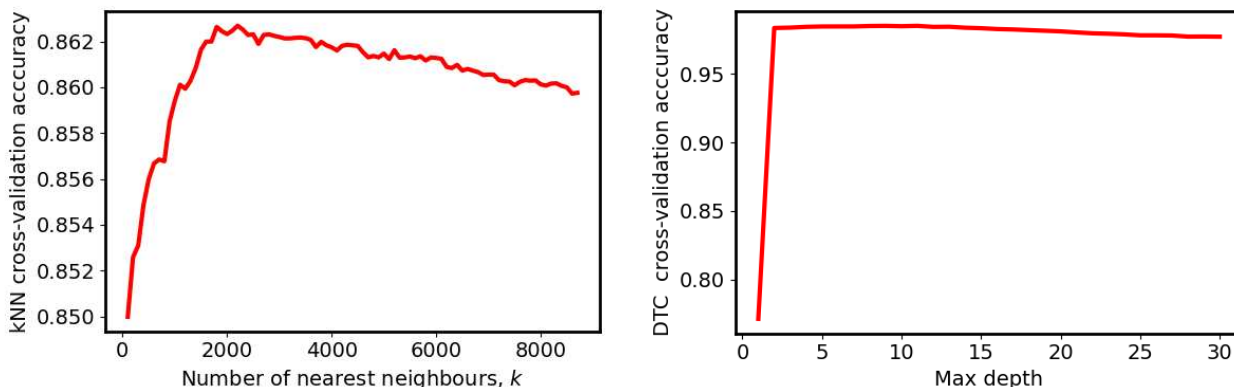


Using `target_classes.py`, the above plot shows the debt-to-income ratio distributions for each status. These overlap to such an extent that any prediction of the status based upon the DTIR alone would essentially be guesswork.

3.2 Model optimisation

Although the SVC scores relatively high, it is far from the highest and very CPU intensive for this size of dataset. As such, the effort required to optimise it is not justified, especially since the must faster DTC yields a score of 98%.

Using `LR.py`, for the LR classifier a grid search was very CPU intensive and failed to complete. Manual experimentation honed in on `C=10`, `solver='newton-cg'` as close to optimal.



Using `kNN.py` and `DTC.py` for the kNN and DT classifiers, the number of nearest neighbours and maximum depths were iterated, giving `n_neighbors ≈ 2000` (a score of 86.2%) and `max_depth = 10` (98.5%),

respectively.

The array to the right shows the confusion matrix for a run of the DT classifier.

	Predicted training		Predicted validation	
	Good	Bad	Good	Bad
Actual good	39 148	136	9747	78
Actual bad	688	39 679	199	9889

The validation data yield 9747 true positives (good loans) and 9889 true negatives (bad loans), compared to 78 false positives and 199 false negatives. This gives a test score of $(9747 + 9889)/(9747 + 9889 + 78 + 199) = 0.986$.

3.3 Feature importance

In order of decreasing positive importance for the LR and DT classifiers:

LR score = 92.414			DTC score = 98.485		
	Feature	Importance		Feature	Importance
18	total_pymnt_inv	2.735509e-01	31	LP_m	0.305927
2	funded_amnt_inv	2.131122e-01	33	debt	0.242985
16	out_prncp_inv	1.798998e-01	16	out_prncp_inv	0.079218
15	out_prncp	1.369901e-01	19	total_rec_prncp	0.030872
19	total_rec_prncp	1.268308e-01	5	installment	0.024627
22	recoveries	1.016987e-01	24	last_pymnt_amnt	0.021162
20	total_rec_int	7.564751e-02	15	out_prncp	0.016005
0	loan_amnt	6.755471e-02	21	total_rec_late_fee	0.014189
5	installment	4.535285e-02	30	issued_m	0.011912
31	LP_m	4.472009e-02	34	DTIR	0.006235
1	funded_amnt	2.436630e-02	4	int_rate	0.003073
33	debt	1.769972e-02	3	term	0.002850
24	last_pymnt_amnt	1.458362e-02	9	inq_last_6mths	0.001361
23	collection_recovery_fee	1.275314e-02	20	total_rec_int	0.001361
21	total_rec_late_fee	1.230367e-02	6	annual_inc	0.001238
32	LCP_m	3.881935e-03	18	total_pymnt_inv	0.000992
34	DTIR	2.013785e-03	7	dti	0.000859
30	issued_m	1.011914e-03	13	revol_util	0.000379
4	int_rate	9.014325e-04	22	recoveries	0.000331
12	revol_bal	3.013145e-04	14	total_acc	0.000254
3	term	1.983654e-04	28	EM_L	0.000171
29	ECL_m	1.757668e-04	17	total_pymnt	0.000141
14	total_acc	1.556792e-04	8	delinq_2yrs	0.000126
10	open_acc	1.556792e-04	36	grade_B	0.000118
74	ILS_f	1.205258e-04	12	revol_bal	0.000118
75	ILS_w	1.205258e-04	2	funded_amnt_inv	0.000110
28	EM_L	1.155039e-04	45	VS_Not Verified	0.000098
56	AS_NC	1.054601e-04	29	ECL_m	0.000095
46	VS_Source Verified	7.532862e-05	23	collection_recovery_fee	0.000093
72	AS_NH+	6.528480e-05	42	HO_MORTGAGE	0.000078
25	collections_12_mths_ex_med	6.277385e-05	1	funded_amnt	0.000068

7	dti	5.775194e-05	10	open_acc	0.000063
39	grade_E	5.273003e-05	46	VS_Source Verified	0.000045
45	VS_Not Verified	5.273003e-05	56	AS_NC	0.000043
65	AS_LA+	4.519717e-05	63	AS_WA	0.000033
71	AS_SC+	3.766431e-05	52	P_credit_card+	0.000030
69	AS_GA+	3.013145e-05	72	AS_NH+	0.000030
51	P_home_improvement+	3.013145e-05	54	AS_MD	0.000025
55	AS_MI	2.762049e-05	65	AS_LA+	0.000025
53	AS_CA	2.008763e-05	0	loan_amnt	0.000020
40	grade_F	1.506572e-05	64	AS_FL+	0.000018
61	AS_TX	1.506572e-05	41	grade_G	0.000015
62	AS_VA	1.506572e-05	49	P_other+	0.000015
41	grade_G	1.506572e-05	48	P_small_business+	0.000015
70	AS_CT+	1.255477e-05	53	AS_CA	0.000015
73	AS_WY+	1.255477e-05	68	AS_KY+	0.000015
43	HO_RENT+	7.532862e-06	38	grade_D	0.000010
49	P_other+	5.021908e-06	50	P_debt_consolidation+	0.000008
8	delinq_2yrs	2.510954e-06	47	VS_Verified	0.000000
60	AS_PA	2.510954e-06	62	AS_VA	0.000000
54	AS_MD	2.510954e-06	74	ILS_f	0.000000
57	AS_NJ	0.000000e+00	73	AS_WY+	0.000000
26	policy_code	0.000000e+00	11	pub_rec	0.000000
42	HO_MORTGAGE	0.000000e+00	71	AS_SC+	0.000000
48	P_small_business+	-2.220446e-17	70	AS_CT+	0.000000
64	AS_FL+	-2.510954e-06	69	AS_GA+	0.000000
59	AS_OH	-2.510954e-06	25	collections_12_mths_ex_med	0.000000
9	inq_last_6mths	-5.021908e-06	67	AS_AZ+	0.000000
27	acc_now_delinq	-5.021908e-06	66	AS_MO+	0.000000
67	AS_AZ+	-7.532862e-06	26	policy_code	0.000000
37	grade_C	-1.004382e-05	27	acc_now_delinq	0.000000
68	AS_KY+	-1.255477e-05	32	LCP_m	0.000000
38	grade_D	-1.255477e-05	61	AS_TX	0.000000
50	P_debt_consolidation+	-1.757668e-05	44	HO_OWN+	0.000000
66	AS_MO+	-2.259859e-05	60	AS_PA	0.000000
58	AS_NY	-2.259859e-05	59	AS_OH	0.000000
11	pub_rec	-2.259859e-05	58	AS_NY	0.000000
52	P_credit_card+	-2.259859e-05	57	AS_NJ	0.000000
47	VS_Verified	-2.259859e-05	35	grade_A	0.000000
36	grade_B	-2.510954e-05	55	AS_MI	0.000000
13	revol_util	-2.762049e-05	37	grade_C	0.000000
63	AS_WA	-3.515336e-05	39	grade_E	0.000000
35	grade_A	-4.017526e-05	40	grade_F	0.000000
6	annual_inc	-4.519717e-05	51	P_home_improvement+	0.000000
44	HO_OWN+	-1.129929e-04	43	HO_RENT+	0.000000
17	total_pymnt	-6.528480e-04	75	ILS_w	0.000000

Eliminating the least important features for the DT classifier ([ML_feat.py](#)), the retention of just the top two features (table to the right) – months since last payment (LP_m) and outstanding debt (loan_amnt - total_rec_prncp)^a is sufficient to achieve 98% accuracy.

^aThe loan amount minus the principal received to date.

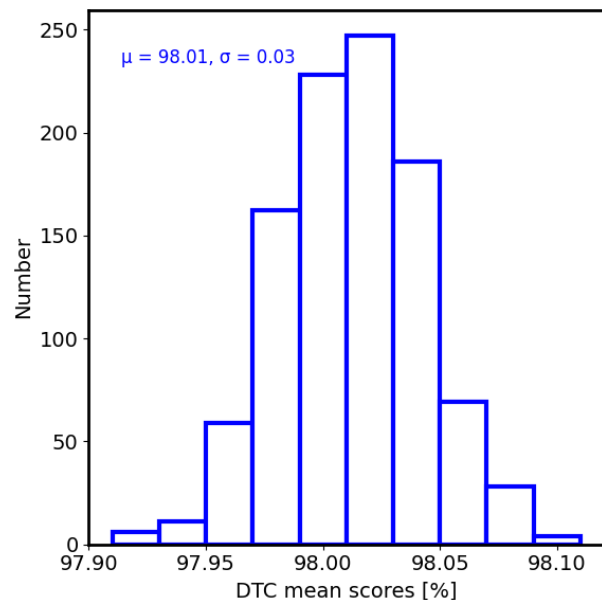
No. of top features	Last feature used	Feature importance	Score [%]
15	annual_inc	> 0.001	98.5
9	issued_m	> 0.01	98.6
6	last_pymnt_amnt	> 0.02	98.3
4	total_rec_prncp	> 0.03	98.1
2	debt	> 0.2	98.0
1	LP_m	–	77
1	debt	–	73

Again, this may be expected since the target value is defined by the mean debt feature normalised by the income. Thus, the model does have some embedded prior knowledge, although, as noted in Sect. 3.1, the spread of debt-to-income ratios for each loan status overlap so much that any direct prediction would not be reliable.

Note that, alternatively, testing the DT classifier with *all* features but LP_m and debt still gives a score 93.2%

It is thus apparent, that with just the *months since last payment*, *loan amount* and *principal received to date*, much of the above processing, including the coarse classing, is unnecessary (at least in this case). This may be evident via the low information values for most of the categorical variables (Sect. 2.2.)

In order to check that the high scores above are not a fluke, although the test sample is relatively large (19913), 1000 trials of the DTC (LP_m and debt only) model were ran, randomising the sample of good =1 from the pool of 416 057 good loans ([ML-loop_feat.py](#)) each time.

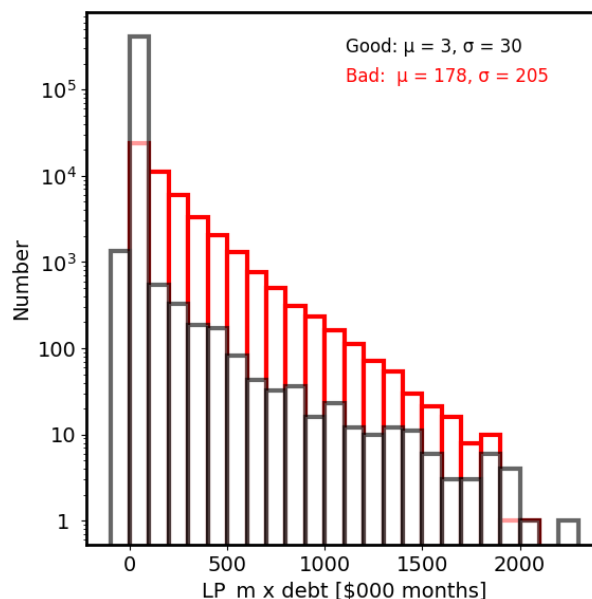


From the resulting distribution of scores, shown by the histogram, it is clear that the score is in fact very stable, as would be expected from a random selection of nearly 20 000.

Given that the DT classifier can predict whether loan is good or bad with $\geq 73\%$ accuracy using either LP_m or debt alone, reaching 98% when both are used, it might be possible that a combination of these features shows a clear distinction between the good and bad loans.

Plotting the products ([2-feat_histo.py](#)), it is seen that the distributions are far from normal preventing us from performing a *t*-test. However, it is clear that the populations mostly overlap, with a large spread in the values.

Thus, it appears as though the machine learning is doing something more complex than a straightforward combination of these two features.



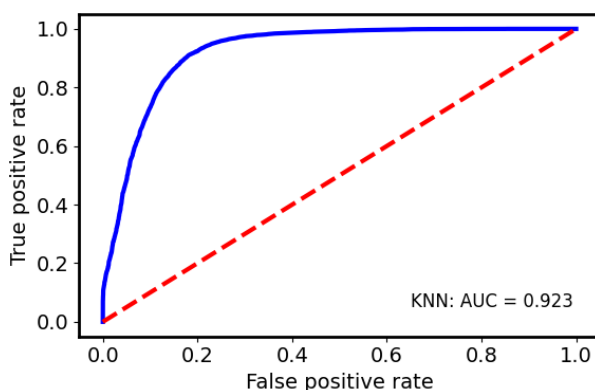
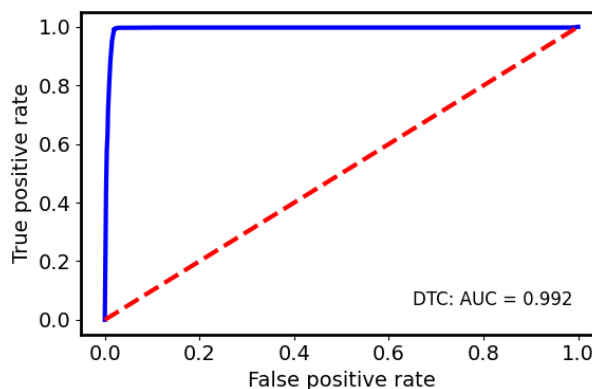
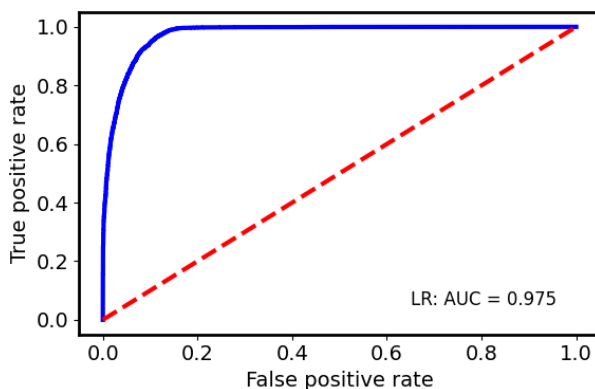
4 Results

4.1 ROC curve

The *receiver operating characteristic* (ROC) is a plot of the *true positive rate* versus the *false positive rate*, obtained from the confusion matrix at a given classification threshold (e.g. $> 0.5 \Rightarrow \text{good} = 1$, $< 0.5 \Rightarrow \text{good} = 0$).

The area under area under the ROC curve (AUC) provides a measure of the effectiveness of the classification. The scores are grouped into categories in the table to the right. For example, predicting purely by chance gives the $\text{AUC} = 0.5$, shown by the red broken lines in the following plots.

Area under ROC curve	Model Interpretation
0.5 – 0.6	No discrimination
0.6 – 0.7	Poor classifier
0.7 – 0.8	Fair classifier
0.8 – 0.9	Good classifier
0.9 – 1	Excellent classifier



The DT classifier gives an $AUC = 0.994$, scoring much higher the 0.5 expected from chance and significantly higher than Udemý's [Credit Risk Modelling in Python](#) AOC of 0.702. *"In credit risk, an AUC of 0.75 or higher is the industry-accepted standard and prerequisite to model acceptance"* (from [Credit Scoring Series Part Five](#)).

4.2 Gini coefficient

The Gini coefficient measures the inequality between the good = 1 and good = 0 borrowers. This is obtained from the AUC, via

$$\text{Gini} = (2 \times \text{AUC}) - 1,$$

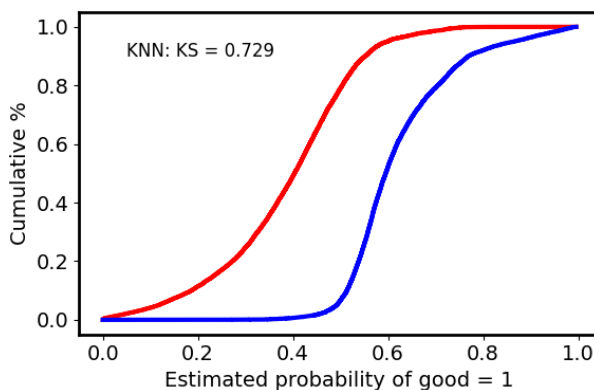
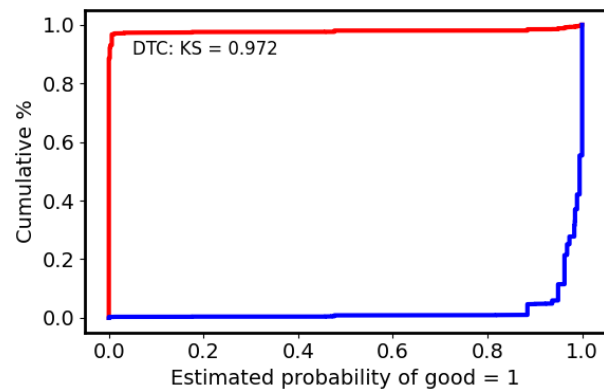
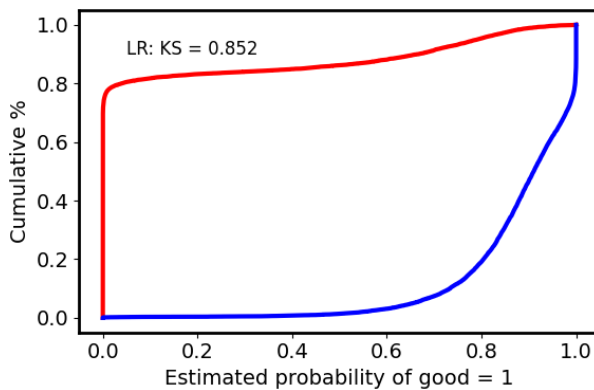
so that the higher the Gini coefficient the better the model in distinguishing the two classes.

The above AUCs give Gini = 95% for LR, 98% for DTC and 85% for kNN, compared to 40%, from the same data, by [Udemý](#) and 71% by [Credit Risk Modelling \(Part II\)](#).

4.3 Kolmogorov-Smirnov score

The Kolmogorov-Smirnov (KS) test compares the distribution of a borrower's credit score with the reference distribution. It uses the maximum difference between the cumulative distribution functions to determine whether the two distributions are significantly different from each other.

We show the KS scores below for the LR and DT classifiers, where the blue trace represents the good = 1 borrowers and the red trace the good = 0 borrowers.



The KS scores of 0.73 – 0.71 are high, indicating that the models have excellent predictive power. Note that both [Udemy](#) and [Credit Risk Modelling \(Part II\)](#) obtain a score of just 0.30.

4.4 Using the Udemy Target Classification

Given the results are much superior to those of [Udemy](#), in order to check that these are not due to the definition of the target (Sect. 2.1.5), we repeat the analysis using the Udemy classification.

Proceeding with `pre-process_Udemy_1.py`, the two classification schemes are shown below.

```
===== my_good = 1 =====
['Fully Paid' 'Current'
 'In Grace Period' 'Late (16-30 days)'
 'Does not meet the credit policy. Status:Fully Paid'
 'Does not meet the credit policy. Status:Charged Off']
===== my_good = 0 =====
['Charged Off' 'Default' 'Late (31-120 days)']
```

Giving 416002 good loans (good = 1)
and 49837 bad loans (good = 0)

```
mean      0.893017
std       0.309092
min       0.000000
max       1.000000
```

```
===== Udemy good = 1 =====
['Fully Paid' 'Current'
 'In Grace Period' 'Late (16-30 days)'
 'Does not meet the credit policy. Status:Fully Paid']
===== Udemy good = 0 =====
['Charged Off' 'Default' 'Late (31-120 days)'
 'Does not meet the credit policy. Status:Charged Off']
```

Giving 415256 good loans (good = 1)
and 50583 bad loans (good = 0)

```
mean      0.891415
std       0.311118
min       0.000000
max       1.000000
```

These are similar, with only `Does not meet the credit policy. Status:Charged Off` being classed differently.

The previous steps (Sects. 2.1.2 – 2.2.7) are repeated for the Udemy classification, with the coarse classing done in `pre-process_Udemy_2.py` and the machine learning in `ML_Udemy.py`, where a randomly selected sample of 50 583 good = 1 loans is used to match the number of good = 0 loans.

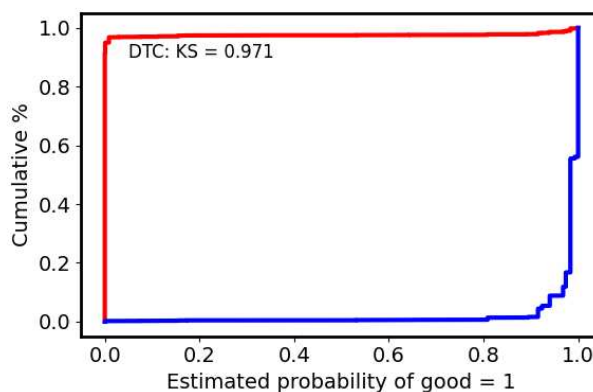
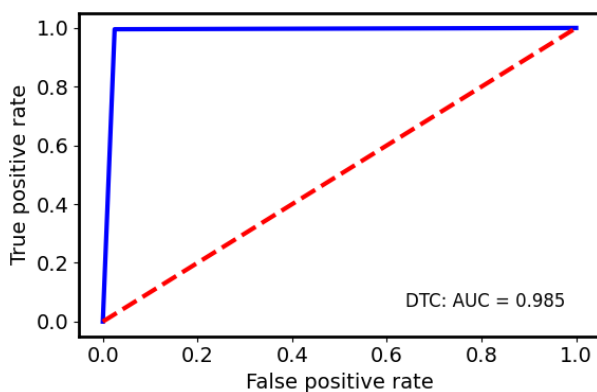
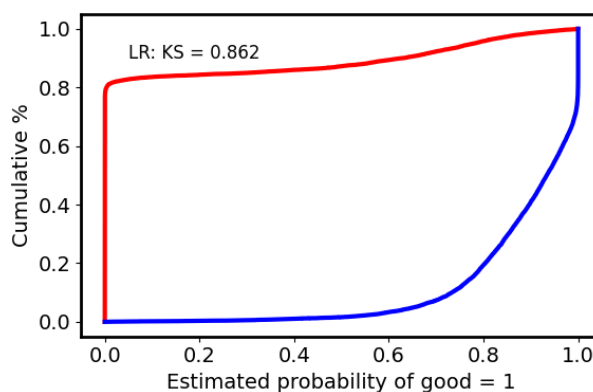
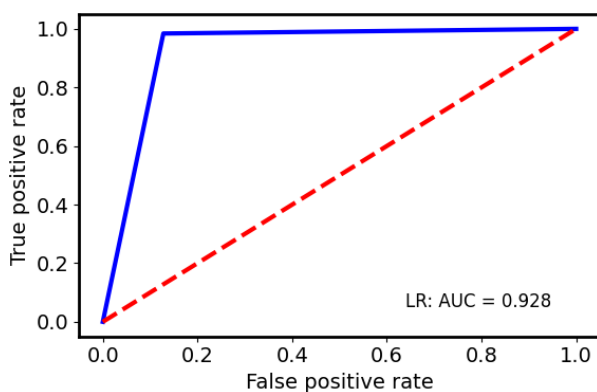
A typical run of the various classifiers gives

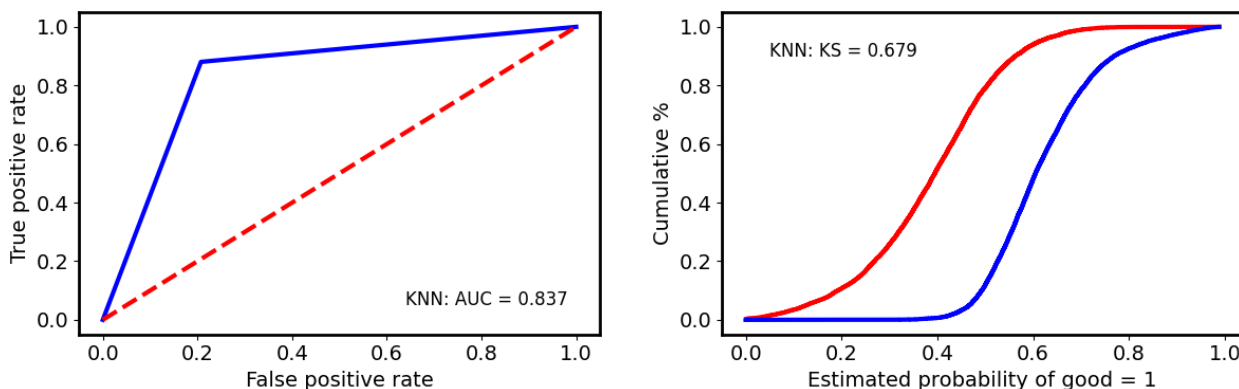
For a 0.2 test fraction (80932 train & 20234 test) LR score = 92.762

For a 0.2 test fraction (80932 train & 20234 test) KNN score = 83.966

For a 0.2 test fraction (80932 train & 20234 test) DTC score = 98.563

which are very similar scores to those above. Therefore, the slightly different classification of loan status is not responsible for the high scores.





4.5 IVs of the Most Important Features

It has been shown above that retaining the continuous features results in a much superior predictive model, with a Gini coefficient of 98% (cf. 40% with fine classing) and a KS score of 0.97 (cf. 0.30).

Furthermore, at least for these data, the categorical features make a relatively small contribution with the decision tree classifier, which still attains accuracies of 98% from only two features – the months since the last payment (LP_m) and outstanding debt (debt).

```
===== Months since last payment (LP_m) =====
```

	LP_m	n_obs	%n_obs	n_good	n_bad	%good	%bad	%n_good	%n_bad	WoE	IV
0	0.000000	179600	38.55	178818	782	99.56	0.44	42.98	1.57	3.31	1.370671
1	1.018501	61998	13.31	61147	851	98.63	1.37	14.70	1.71	2.15	0.279285
2	2.004148	9317	2.00	7411	1906	79.54	20.46	1.78	3.82	-0.76	0.015504
3	3.022649	11248	2.41	8975	2273	79.79	20.21	2.16	4.56	-0.75	0.018000
...											
95	94.983470	7	0.00	7	0	100.00	0.00	0.00	0.00	NaN	NaN
96	96.001971	9	0.00	9	0	100.00	0.00	0.00	0.00	NaN	NaN
97	97.020473	2	0.00	2	0	100.00	0.00	0.00	0.00	NaN	NaN

```
=====
```

For 'LP_m' feature, sum of IVs = 2.1116: IV > 0.5 - something fishy going on

```
=====
```

```
===== Outstanding debt in $000 (debt_k) =====
```

	debt_k	n_obs	%n_obs	n_good	n_bad	%good	%bad	%n_good	%n_bad	WoE	IV
0	-0.6	1	0.00	1	0	100.00	0.00	0.00	0.00	NaN	NaN
1	0.0	185020	39.72	184999	21	99.99	0.01	44.47	0.04	7.01	3.114543
2	0.1	387	0.08	343	44	88.63	11.37	0.08	0.09	-0.12	0.000012
3	0.2	713	0.15	666	47	93.41	6.59	0.16	0.09	0.58	0.000406
...											
348	34.8	1	0.00	0	1	0.00	100.00	0.00	0.00	NaN	NaN
349	34.9	1	0.00	0	1	0.00	100.00	0.00	0.00	NaN	NaN

```
350    35.0      2    0.00      0      2    0.00  100.00      0.00      0.00    NaN      NaN
```

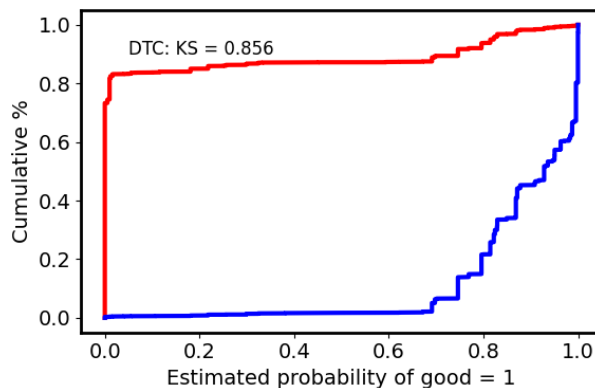
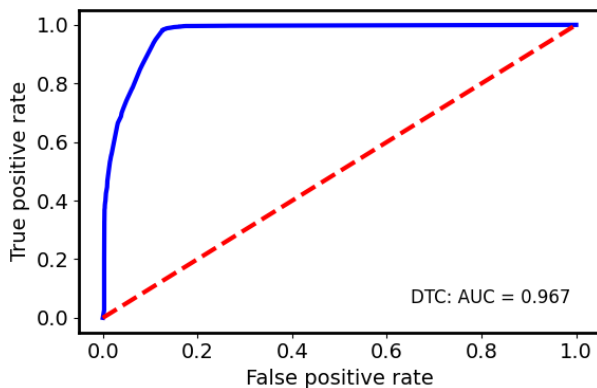
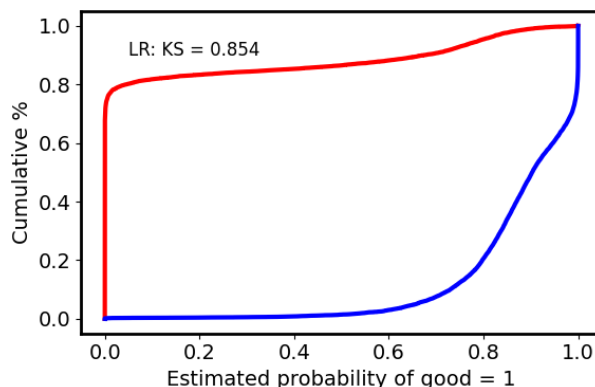
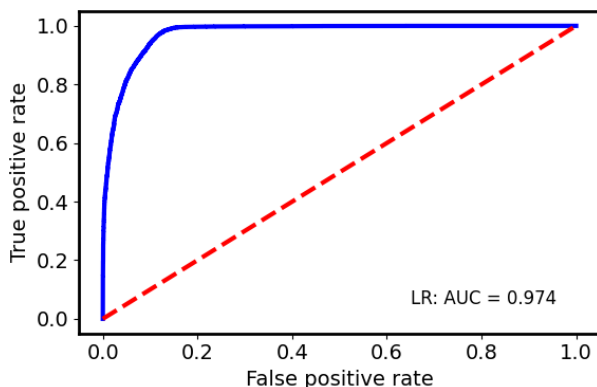
```
=====
For 'debt_k' feature, sum of IVs = 3.4508: IV > 0.5 - something fishy going on
=====
```

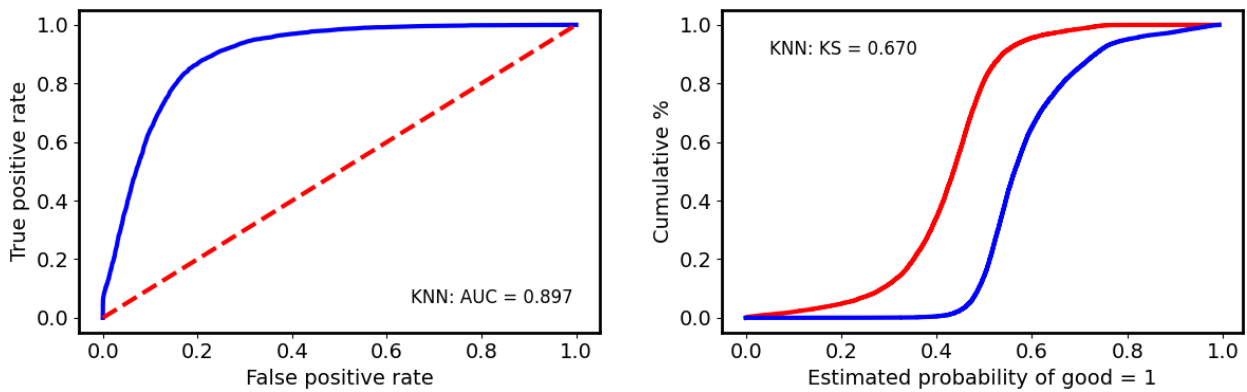
Looking at the information values, both are high, $IV \gg 0.5$, which suggests leakage. That is, they are acting as a proxy for the prediction, although, as seen from above, the relationship is not simple.

As mentioned in Sect. 3.3, however, the DT classifier maintains a 93% accuracy when all the features *except* these two are used. This is most likely due to the richness of the dataset which still comprises 74 features. In order to test if the above is the result of leakage, in `ML_all-others.py` we remove these features as well as the debt-to-income ratio and the annual income:

```
del df['LP_m']; del df['debt']; del df['DTIR']; del df['annual_inc']
```

The cross-validation score remains high, at 93%, and the Gini coefficient is 0.94 with a KS score ≥ 0.85 for both the LR and DT classifiers.





That is, high scores are still attained without the features used to define the target, suggesting that the results are not dominated by leakage.